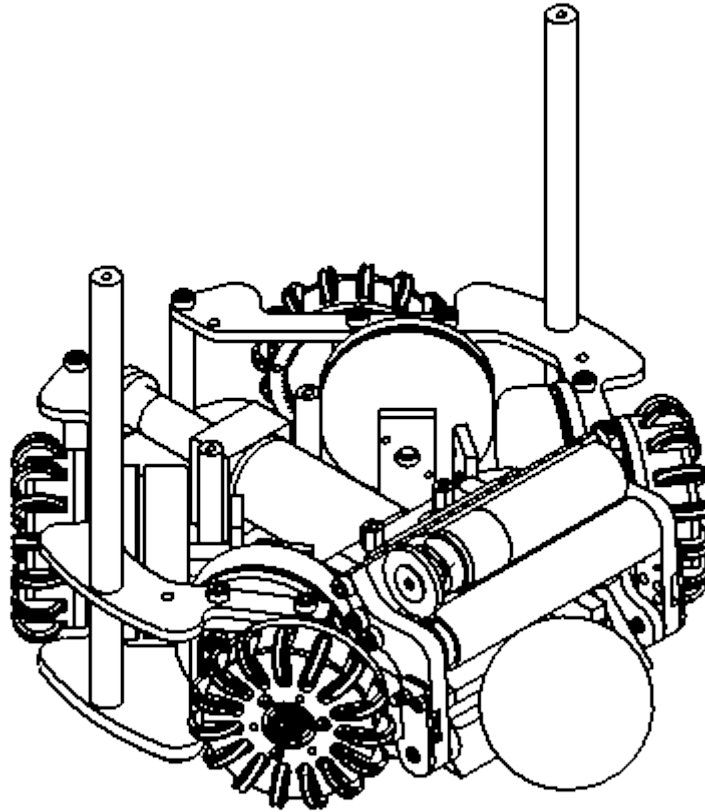


# RAPPORT DE PROJET



## Projet Robocup

### Responsable Pédagogiques :

Pierre AUBRY

Aloïs MICLO

### Intervenant :

Steve N'GUYEN

## Table des matières

<b>PRESENTATION DU PROJET .....</b>	<b>2</b>
Contraintes mécaniques et électroniques .....	2
Contraintes économiques.....	2
Contraintes logicielles .....	2
<b>WHAT DID I HAVE TO DO ? .....</b>	<b>3</b>
Diagramme de Gant.....	3
Cahier des charges .....	4
Centrale inertielle .....	4
Capteur optique.....	5
Descriptions des capteurs.....	5
Centrale inertielle .....	5
Capteur optique.....	6
Capteur angulaire .....	7
<b>ANALYSE FONCTIONNELLE.....</b>	<b>7</b>
Schéma fonctionnel IMU .....	7
Schéma fonctionnel capteur optique.....	8
Schéma fonctionnel capteur angulaire .....	8
<b>MICROCONTROLEUR .....</b>	<b>8</b>
Analyse fonctionnelle .....	9
Analyse structurelle .....	10
Schéma structurel .....	10
Analyse structurelle.....	10
Fabrication .....	13
Nomenclature .....	15
Tests & validations.....	15
<b>ETUDE LOGICIELLE .....</b>	<b>16</b>
Fonction I <sup>2</sup> C .....	16
Fonction SPI.....	18
Fonction UART.....	21
Fonction Timer .....	24
PWM .....	24
Output Compare.....	25
Asservissement en vitesse .....	26
Déplacement & Position.....	27
Détermination des équations de déplacements.....	27
Calcul de la position du robot.....	29
<b>CONCLUSION .....</b>	<b>31</b>
<b>ANNEXE.....</b>	<b>31</b>

## PRESENTATION DU PROJET

L'objectif du projet est de réaliser un robot holonome pour la ligue SSL de la Robocup.

La ligue SSL de la Robocup réunit des équipes de 6 robots s'affrontant en match de football.

Les robots sont holonomes et contrôlent via un pc distant. Ce dernier a accès à des caméras visionnant le terrain, et peut ainsi coordonner les robots pendant le match. Nous devons alors réaliser un premier prototype à perfectionner en vue de la Robocup 2020 se tenant à Bordeaux.

Afin de réaliser plusieurs prototypes en même temps, nous avons été repartis en équipe de 4. Ci-dessous, la liste des membres de mon groupe, ainsi que la tâche que chacun a eu à réaliser :

Chargé de la fonction	Fonction à réaliser
Joris OFFOUGA	Gestion moteur & déplacement du robot.
Yassine ELMERNISSI	Gestion RF, alimentation et conception carte mère du robot.
Éric REBILLON	Conception et mise en œuvre du dribleur, mise en œuvre du solénoïde pour tirer.
Matthieu CASTETS	Conception d'ESC <<home made>>, meca du robot et mise en œuvre des moteurs.

## Contraintes mécaniques et électroniques

Le robot SSL sera capable de :

- Tirer à une vitesse de l'ordre de 6m/s,
- Dribler à l'aide d'un rouleau capable de bloquer la balle pour effectuer un tir,
- Et le contrôle des roues sera assuré à l'aide de moteurs BRUSHLESS équipés de capteur à effet hall

Il sera de forme cylindrique, et aura 25 cm maximum de diamètre et de hauteur. Il sera alimenté en 12v par une batterie LIPO 3S, et la gestion de charge ainsi que l'état de la batterie sera effectuée à bord de l'appareil.

## Contraintes économiques

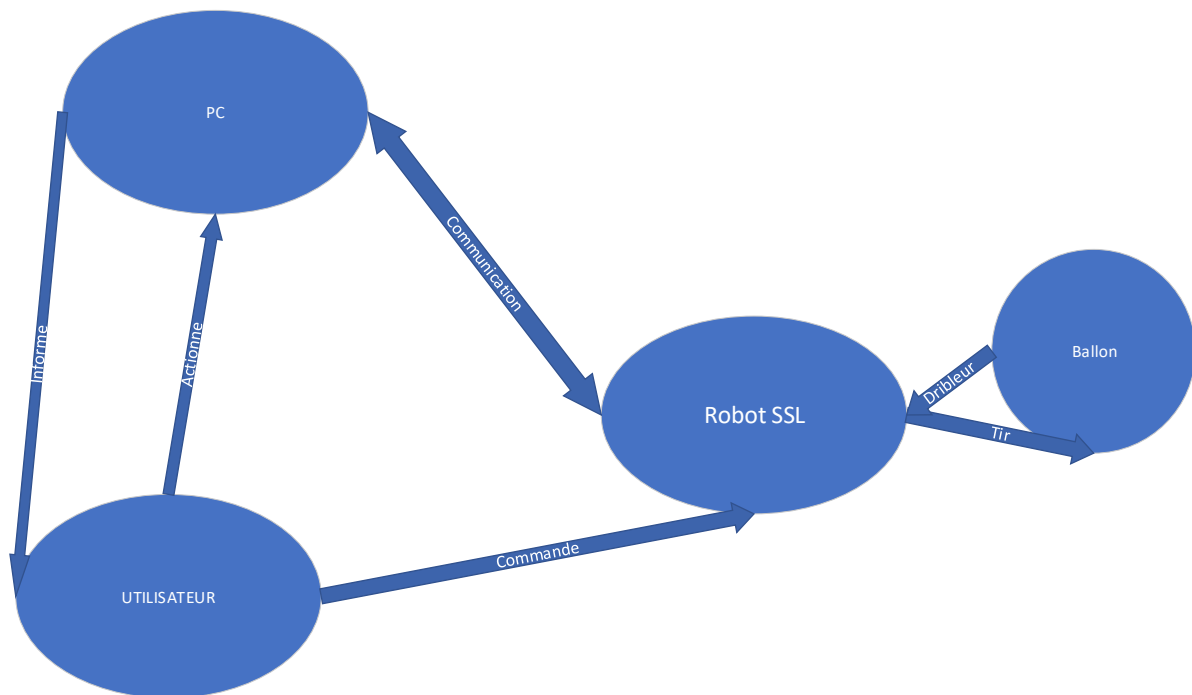
Pour ce projet chaque équipe dispose d'un budget de 500 €.

## Contraintes logicielles

Pour le développement logiciel du robot, il devra être équipé :

- D'une centrale inertielle ainsi que de deux capteurs optiques de type souris pour assurer un asservissement de robot,
- D'un système de détection de la balle agissant avec le dribleur pour permettre au robot de tirer,
- Et la communication sans fil avec l'interpréteur de commande du robot à savoir un pc se fera à l'aide d'un module radio nRF24L01.
- Afin d'effectuer un contrôle en vitesse, un asservissement devra être effectué à l'aide de capteur angulaire placé sur les roues.

Afin de rapidement présenter le comportement du robot, voici un diagramme sagittal :

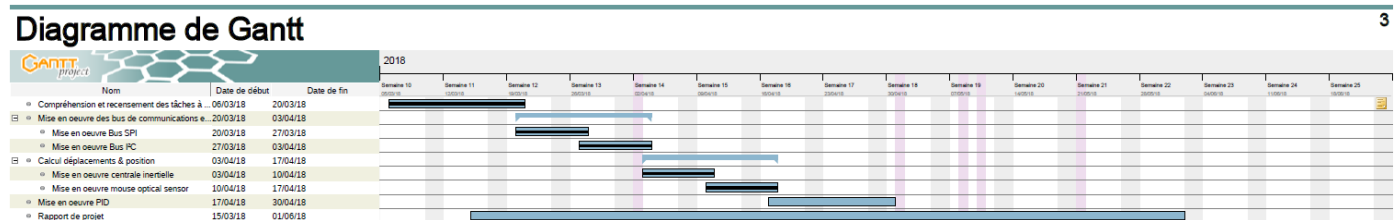


Voici donc une présentation globale du projet à réaliser. Nous allons maintenant nous intéresser à la partie que j'ai eu à réaliser au cours de celui-ci.

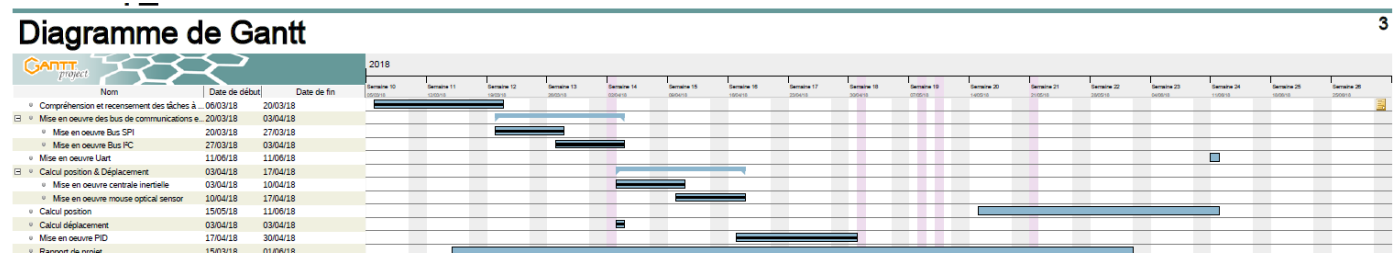
## WHAT DID I HAVE TO DO ?

### Diagramme de Gantt

Gantt prévisionnel



Gantt réel






## Cahier des charges

Au vu des différentes tâches à accomplir, j'ai eu à examiner différents types de capteurs pour respecter le cahier des charges.

### Centrale inertielle

Pour la centrale inertielle, il nous est demandé qu'elle contienne un accéléromètre et un gyroscope. Au vu de cela, j'ai alors cherché une IMU qui contienne les deux.

Critères	Initial	Personnel
Alimentation	NC	3,3V
Degrés de liberté	NC	>6°
Résolution ADC	NC	16 bits
Axes	NC	3 axes
Plage de mesure	NC	16±g & 2000°/s
Interface	NC	I <sup>2</sup> C
Sensors	Accéléromètre & Gyroscope	Accéléromètre & Gyroscope
Courant	NC	<20mA
Prix	NC	<20€

Module	Interface /MCU	Capteurs	Précision	Prix	Image
GY-525 MPU6050	I <sup>2</sup> C/None	Accéléromètre, Gyroscope (3 axes)	ADC 16 bits max	1,42 €	
MPU9050	I <sup>2</sup> C / None	Accéléromètre, Gyroscope, Magnétomètre (3 axes)	ADC 16 bits max 9 Degrés de liberté	7,99 €	
Razor IMU	UART / ARM Cortex M0	Accéléromètre, Gyroscope, Magnétomètre (3 axes)	ADC 16 bits max 9 Degrés de liberté	60,60 €	

En prenant en compte la partie de mes camarades, et ayant déjà commencé à utiliser le module GY-525, j'ai donc décidé d'utiliser celui-ci pour le projet. Ce module est utilisé dans de nombreux projets robotiques, pour son faible coût, mais surtout pour sa facilité de mise en œuvre. Cependant la Razor IMU qui est un module à base de MPU9050 et surtout couplé à un ARM cortex M0, permet de restituer directement des angles d'eulers. Il aurait été un choix plus judicieux, mais le budget ne le permettait pas. Et le fait qu'il contienne un microcontrôleur d'une architecture et d'une famille différente, ne lui permettait pas de respecter le cahier des charges.

## Capteur optique

Pour le choix du capteur optique, étant pourtant utilisé dans certaines souris, était introuvable chez les fournisseurs professionnels comme Farnel et Mouser. En étoffant mes recherches, j'ai pu trouver un module chez Amazon, le CJMCU, qui contient un circuit ADNS3080 conçu par Avago, à un prix de 15,66 €.



Pour le capteur angulaire, le choix revient à mon camarade qui s'occupe de la partie ESC.

Voici donc un récapitulatif des capteurs que j'aurai à utiliser et des tâches à réaliser avec :

- Capteur de type souris ADNS3080
- Centrale inertielle MPU6050
- Mise en œuvre d'une carte microcontrôleur avec IMU et capteur de type souris
- Capteur angulaire AS5048A
- Calcul du PID en vitesse et position du robot
- Calcul des déplacements du robot

## Descriptions des capteurs

### Centrale inertielle

Une centrale inertielle est un circuit électronique composé en général de deux capteurs, un accéléromètre et un gyroscope. Mais il est courant de les coupler avec un magnétomètre pour corriger les erreurs de mesure.

#### Accéléromètre

Un accéléromètre est un capteur permettant de mesurer les accélérations. Il repose sur le principe de la formule :  $\vec{F} = m\vec{a}$ .

En effet, ce capteur renvoi une mesure d'accélération liée aux chocs ou aux mouvements.

Il est possible en réalisant une intégration par rapport au temps d'obtenir une vitesse mais cette dernière dérivera dans le temps :

$$v = \int a. dt$$

Il en va de même pour obtenir un déplacement, on procédera à une double intégration :

$$d = \iint a. dt$$

Cependant, effectuer ce genre d'opération sur la donnée brute du capteur ne ferait qu'amplifier le bruit de la mesure. Pour éviter cela il faut combler les opérations avec le gyroscope.

Pour utiliser le gyroscope il nous faut d'abord obtenir les angles d'inclinaison.

$$\theta = \tan^{-1} \frac{Ay}{\sqrt{Ax^2 + Az^2}}$$

$$\varphi = \tan^{-1} \frac{Ax}{\sqrt{Ay^2 + Az^2}}$$

$$\psi = \tan^{-1} \frac{\sqrt{Ax^2 + Az^2}}{Az}$$

### Gyroscope

Le gyroscope permet de mesurer des vitesses de rotations même très faible. En termes de mesures, il est plus fiable que l'accéléromètre.

Il nous faut intégrer les vitesses de rotations pour obtenir les angles :

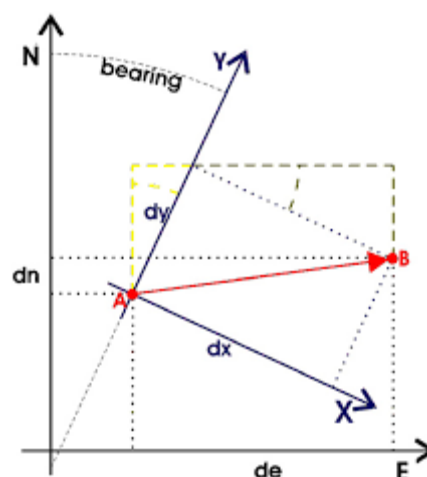
$$\theta = \int gx \cdot dt ; \varphi = \int gy \cdot dt ; \psi = \int gz \cdot dt$$

Nous pouvons obtenir donc plusieurs données pour les angles du robot. Cependant, ces angles contiennent énormément de bruit ou sont totalement biaisés, il nous faudra donc appliquer un filtre et faire de la fusion de données pour combler les erreurs de mesure et ainsi avoir une estimation correcte de sa position à travers ses angles.

### Capteur optique

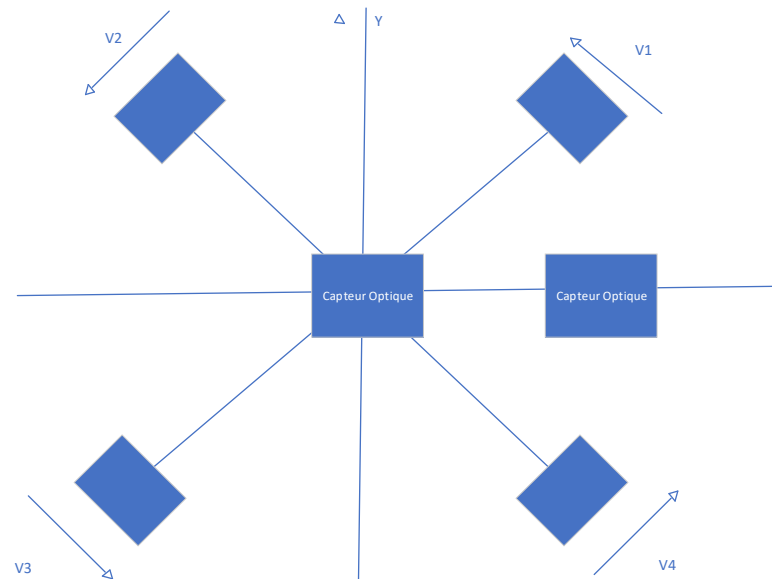
Le capteur ADNS3080 est un capteur utilisé surtout pour les souris, en effet, il permet de récupérer des variations de vitesse suivant les axes x et y.

Il peut être utilisé alors pour calculer des vitesses  $v = \sqrt{dx^2 + dy^2}$ , des accélérations par dérivées  $a = \frac{dv}{dt}$  et même réaliser un parcours :



On utilisera alors la formule  $de = dy \cdot \sin(\text{bearing}) + dx \cdot \cos(\text{bearing})$

Voici la disposition des capteurs optiques sur le robot :



Le capteur optique au centre servira à la détermination de la rotation et de la position, et celui de droite sera utilisé pour la vitesse et la position également.

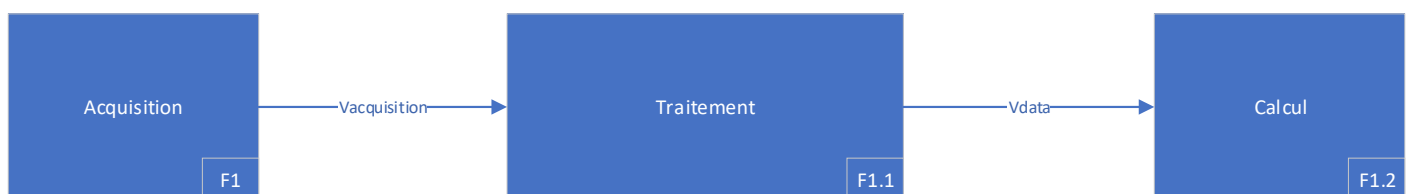
### Capteur angulaire

Le capteur AS5048A peut lire des angles jusqu'à 360°. Il peut donc être utilisé pour le calcul de vitesse d'une roue pour réaliser l'asservissement en vitesse, mais peut aussi être utilisé pour réaliser l'odométrie du robot.

Malheureusement, hormis la documentation du constructeur, je n'ai pas pu utiliser le capteur. Nous voulions le capteur avec une interface de sortie I<sup>2</sup>C. Nous avons passé commande auprès de notre fournisseur, mais le délai de réapprovisionnement était visiblement plus long que celui indiqué sur le site.

## ANALYSE FONCTIONNELLE

### Schéma fonctionnel IMU



Description des fonctions :

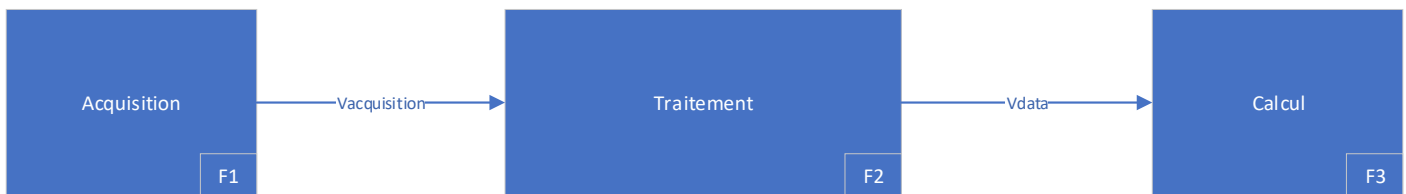
- Acquisition : Acquisition des données brutes par l'IMU
- Traitement : Récupération des données brutes par le MCU et premier traitement
- Calcul : Utilisation des données traitées pour calculer les angles d'eulers.

Descriptions des signaux :

- Vacquisition : Signaux I<sup>2</sup>C contenant les données brutes de l'IMU
- Vdata : Signaux SPI des données traiter sur le PC



## Schéma fonctionnel capteur optique



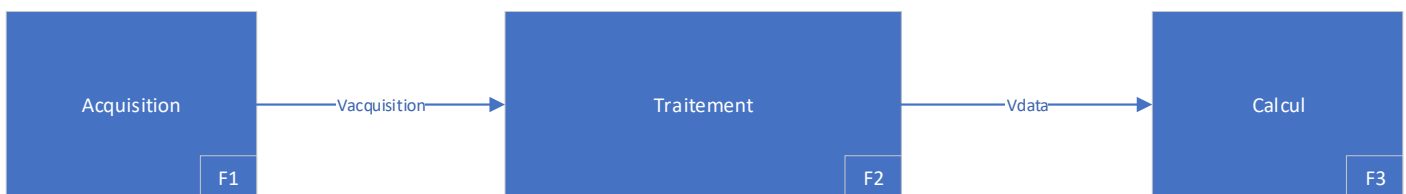
### Description des fonctions :

- Acquisition : Obtention des données brutes par le capteur
- Traitement : Récupération par le MCU et calcul de la vitesse
- Calcul : Comparaison de la vitesse mesurée avec celle des autres capteurs pour asservissement

### Description des signaux :

- Vacquisition : Signaux SPI contenant les données brutes du capteur
- Vdata : Signaux SPI contenant les données à traiter sur le PC

## Schéma fonctionnel capteur angulaire



### Descriptions des fonctions :

- Acquisition : Obtention des données brutes par le capteur
- Traitement : Récupération par le MCU et calcul de la vitesse
- Calcul : Comparaison de la vitesse mesurée avec celle des autres capteurs pour asservissement

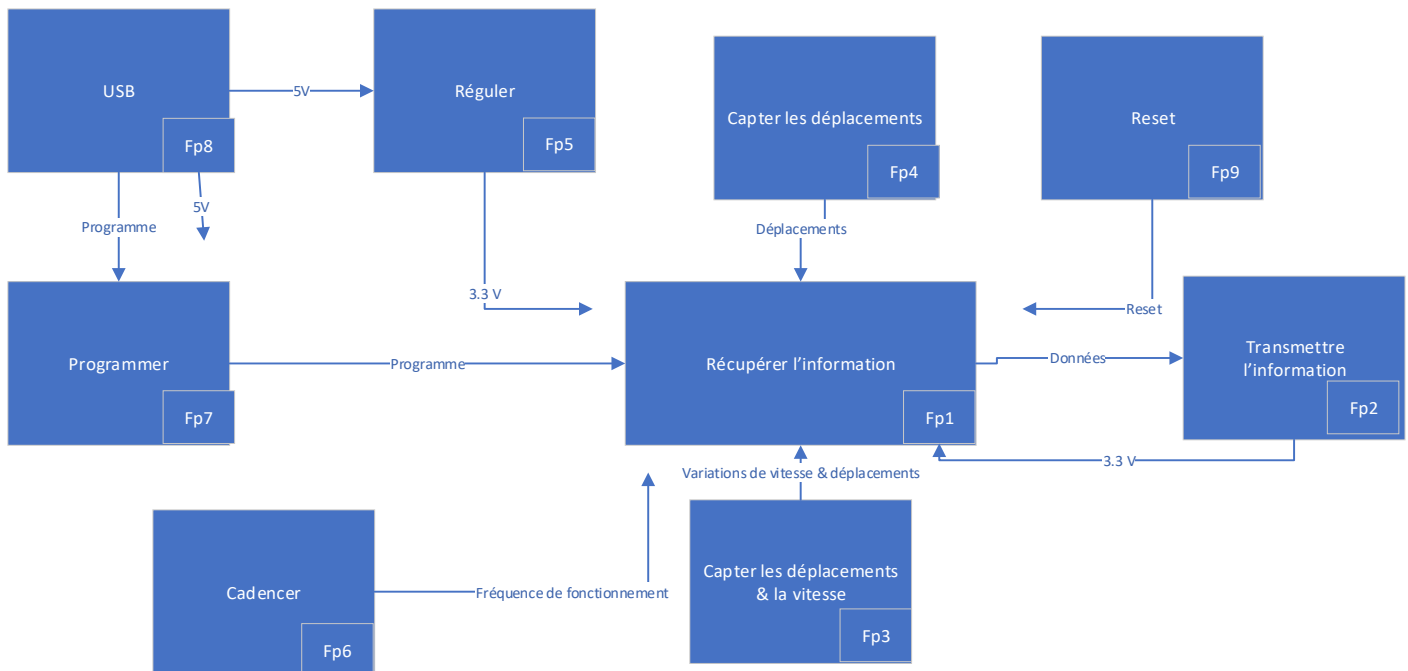
### Descriptions des signaux :

- Vacquisition : Signaux I<sup>2</sup>C contenant les données brutes du capteur
- Vdata : Signaux SPI contenant les données à traiter sur le PC

## MICROCONTROLEUR

Au cours du projet, je me suis rendu compte que nous aurions un problème pour recueillir et traiter toutes les données des capteurs. J'ai donc convenu avec mon camarade qui s'occupe de la partie carte mère qu'il fallait qu'on réalise une carte microcontrôleur de soutien pour la carte mère. J'ai donc pris en charge la conception de cette carte. C'était aussi pour moi l'occasion de renouer avec l'électronique.

## Analyse fonctionnelle



### Description des fonctions :

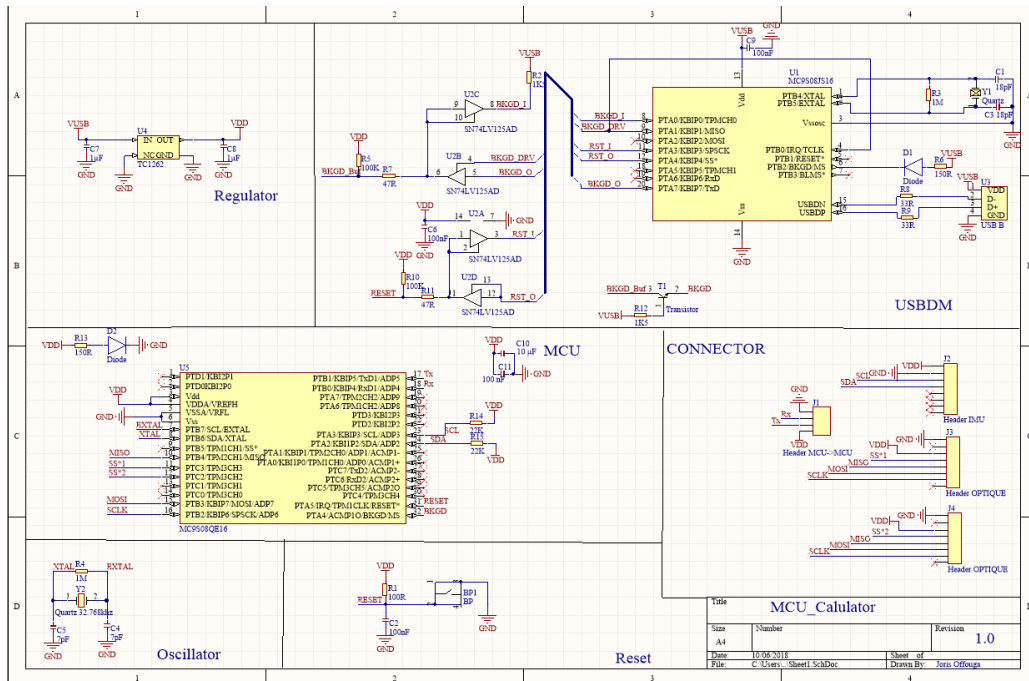
- Fp1 : Microcontrôleur chargé de récupérer les informations des capteurs, possibilité de calculs de position.
- Fp2 : Microcontrôleur chargé de transmettre les données ou les résultats des opérations au pc.
- Fp3 : Informe le microcontrôleur des déplacements et de la vitesse du robot.
- Fp4 : Informe le microcontrôleur des déplacements du robot.
- Fp5 : Permet d'alimenter le microcontrôleur lorsqu'il faut le programmer.
- Fp6 : Cadence le microcontrôleur à la fréquence souhaitée via la PLL.
- Fp7 : Programmeur permettant de transmettre les programmes au microcontrôleur.
- Fp8 : Fonction permettant à l'utilisateur de programmer la carte.
- Fp9 : Permet de remettre à zéro les registres et relance le programme au départ.

### Description des signaux

- 3,3 V : Alimentation du microcontrôleur et des capteurs.
- 5 V : Alimentation du programmeur
- Déplacements : Signaux I<sup>2</sup>C contenant les données brutes du capteur.
- Programme : Binaire contenant les programmes.
- Reset : Signal TOR pour remettre à zéro le microcontrôleur.
- Variations de vitesse & déplacements : Signaux SPI contenant les données du capteur.
- Données : Signaux UART contenant les données à transmettre.
- Fréquence de fonctionnement : Fréquence de fonctionnement.

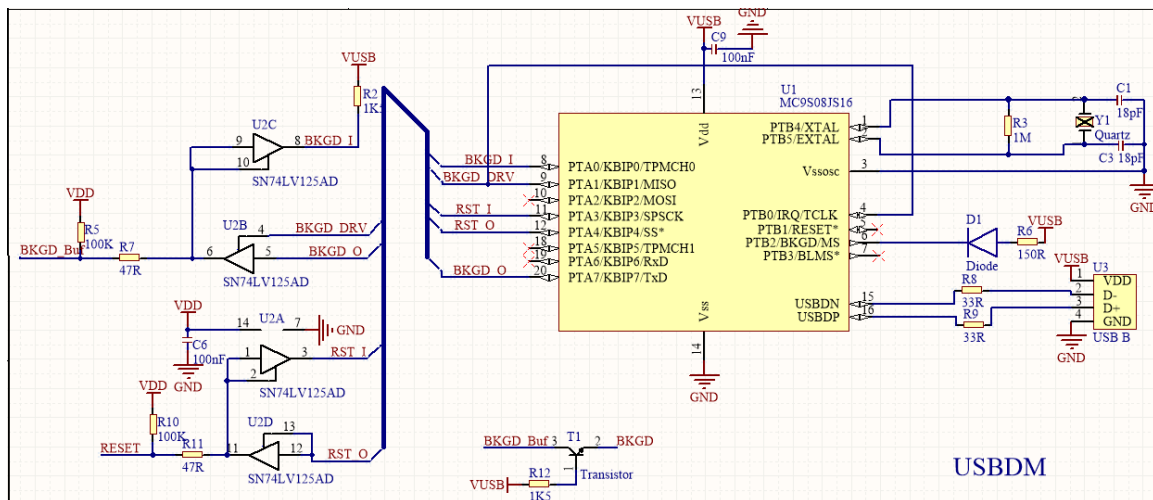
## Analyse structurelle

### Schéma structurel



## Analyse structurelle

### USBM

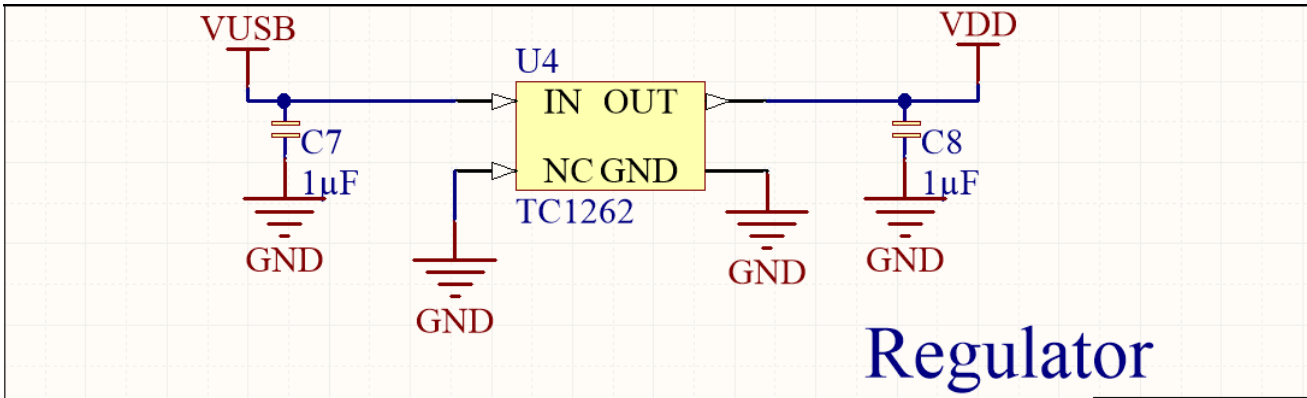


Pour la partie en charge de la programmation du Microcontrôleur, je me suis référé au projet open source USBDM héberger sur sourceforge : [USBDM](#). Il regroupe un choix de bootloader pour les microcontrôleurs de la famille NXP. Etant partie sur un HSC08, je suis allé trouver la schématique sur GitHub : [USBDM\\_HCS08](#). Expliquons donc comment fonctionne l'USBDM.

Un microcontrôleur (Programmateurs) qui contiendra le bootloader pour programmer le microcontrôleur principal. L'environnement de ce programmeur contient un quadripôle buffer le SN74LV125AD permettant l'adaptation en courant de plusieurs signaux venant d'un système 5V vers un autre système 5V ou bien 3,3V. Un quartz à 12 MHz pour le fonctionnement du programmeur, une ILM pour montrer la présence du mode debug ou bien du téléversement du code vers le microcontrôleur principal et d'un USB type B pour la communication avec le pc. Enfin

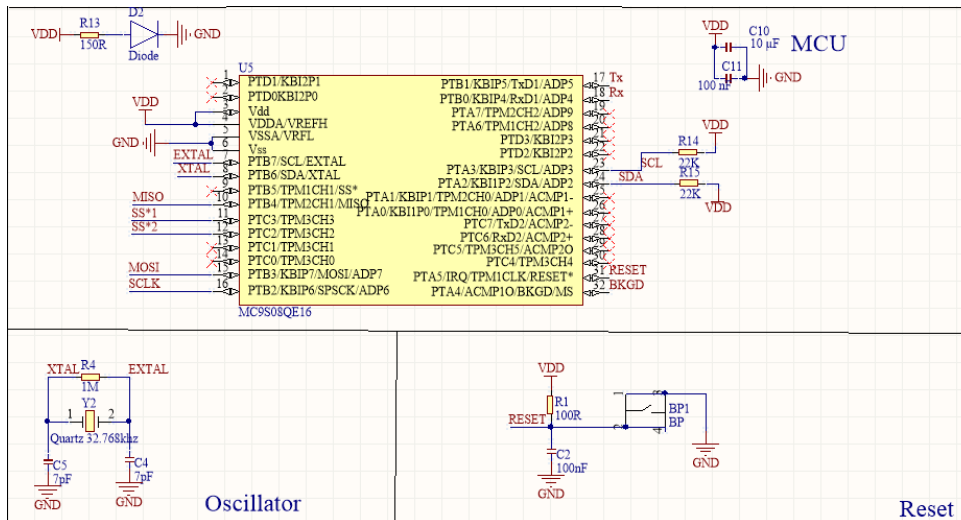
le transistor, maintient la broche BKGD du microcontrôleur à l'état haut, car le programmeur lui, la force à l'état bas quand il n'est pas alimenté, ce qui a pour effet de réinitialiser le microcontrôleur et nous empêche de le programmer.

### Régulateur



Le TC1262 est un régulateur 3,3V. Son environnement se compose de deux capacités de découplage. Il délivre au minimum 500 mA. Ce régulateur ne servira qu'alimenter le microcontrôleur que lorsqu'on branche l'USB pour programmer la carte.

### MCU



Pour soutenir le microcontrôleur principal, j'ai choisi un MC9S08QE16 aux jeux d'instruction CISC de la famille NXP. Il possède une interface UART, une interface I2C, une interface SPI, 3 Timer, 28 I/O, 10 channels ADC et deux ports comparateurs.

Il dispose d'une IHM pour montrer qu'il est bien alimenté et ainsi que de deux capacités de découplage. Elles sont présentes pour filtrer l'alimentation de celui-ci. La fréquence de coupure de ce filtre est de :

$$F_{C10} = \frac{1}{2\pi * 27,5 * 10 * 10^{-6}} = 578,74Hz$$

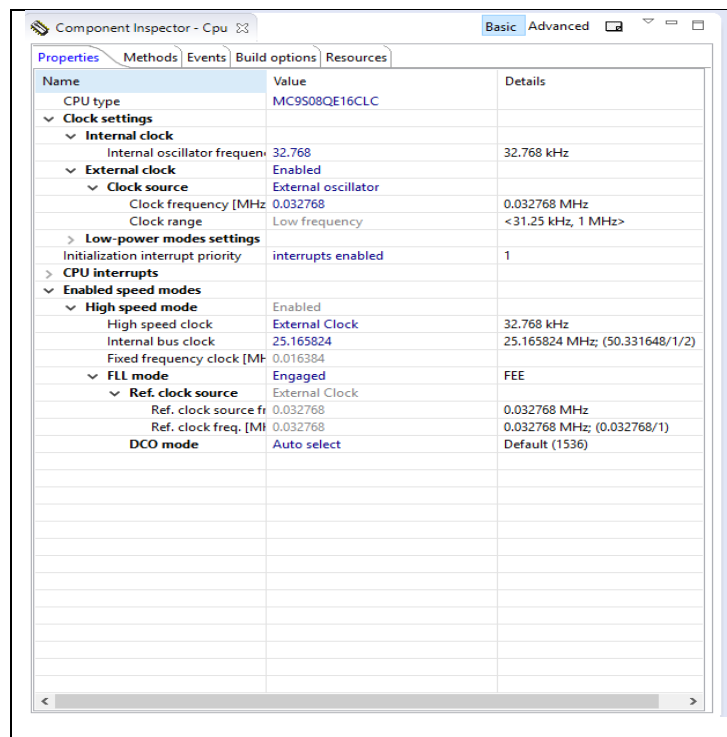
$$F_{C11} = \frac{1}{2\pi * 27,5 * 0,1 * 10^{-6}} = 57874Hz$$

## Oscillateur

L'oscillateur interne du microcontrôleur n'étant pas très précis, nous avons donc choisi d'utiliser un oscillateur à quartz. En effet, celle du microcontrôleur n'est ni plus ni moins qu'une simple cellule RC or pour utiliser des timers et avoir de bon fonctionnement avec des bus de communications, il faut que la fréquence de bus soit précise. C'est pourquoi nous utilisons un oscillateur à quartz. Il s'agit d'une structure colptis, son atout réside dans sa robustesse et sa précision. Le quartz oscille à 32,768 KHz car lors d'une utilisation pour une PLL interne comme nous allons le faire, il sera plus facile grâce à sa facilité de multiplication d'obtenir différentes valeurs de fréquence de bus.

La fréquence de fonctionnement du microcontrôleur sera donc de 50.33MHz avec la PLL, et la fréquence de bus que j'ai choisis pour le cadencement du MCU est de 25.165 MHz.

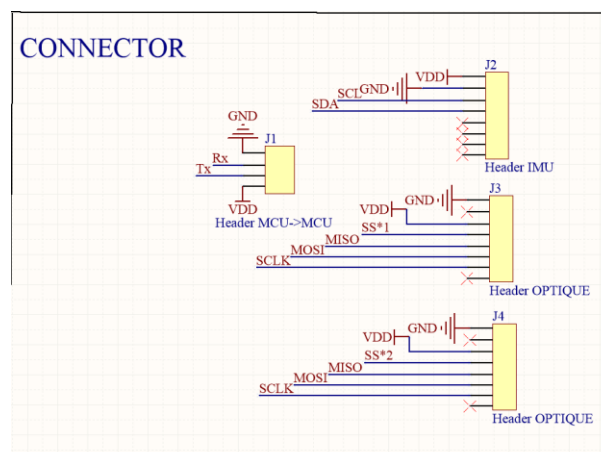
Pour la PLL interne, il faut se rendre sur l'IDE Code Warrior pour l'appliquer. Exemple :



## Reset

La fonction reset du microcontrôleur est à initialiser dans l'IDE par le développeur. Cette fonction nécessite une impulsion de 100ms calculer par la relation  $\tau = RC = 100 * 100.10^{-6} = 100ms$

## Connecteurs



Pour la communication MCU->MCU, nous avons choisi d'utiliser l'UART, n'ayant besoin que de trois signaux à savoir Tx, Rx et GND.

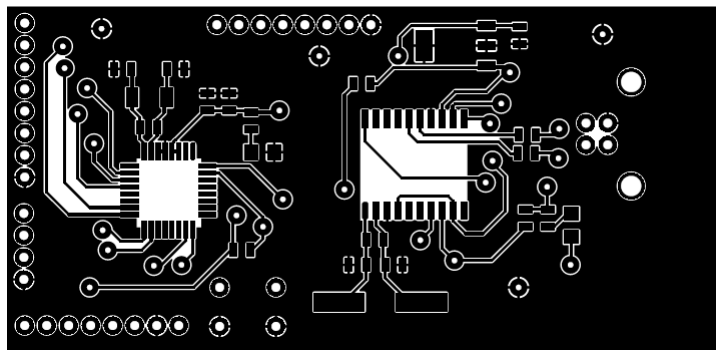
Le connecteur pour l'IMU, nécessite juste les signaux d'alimentation, de référence (ground) et les deux signaux de communications I<sup>2</sup>C.

Pour les capteurs optiques, le connecteur est relié à l'alimentation, la référence et les signaux SPI avec un chip select différent pour chacun.

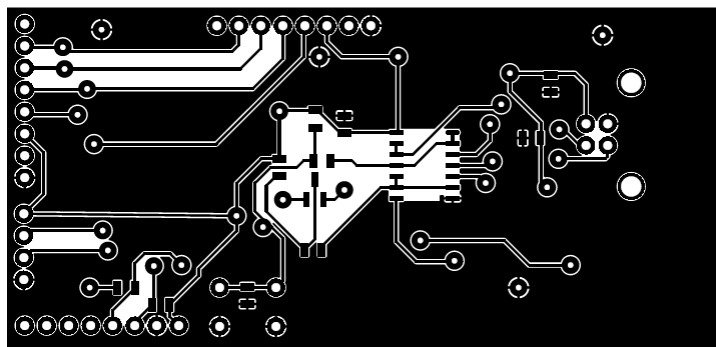
## Fabrication

Typon

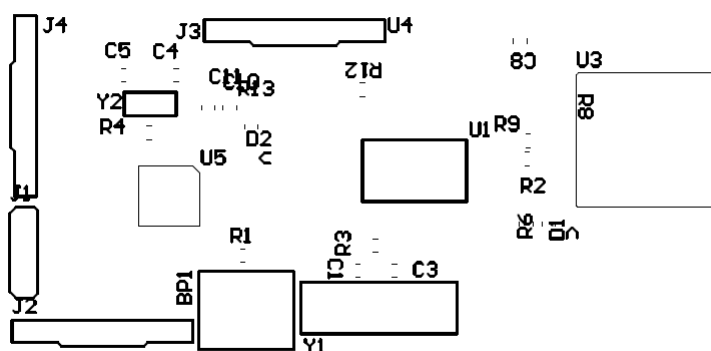
Top Layer

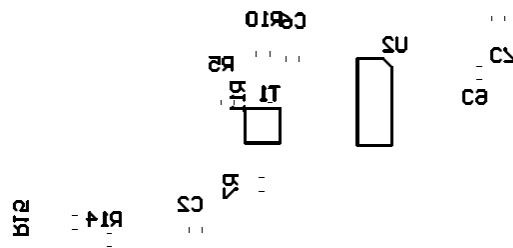


Bottom Layer



Plan de Plaçage





## Nomenclature

Liste des composants nécessaire à la réalisation de la carte

Comment	Description	Designator	Footprint	LibRef	Quantity
BP		BP1	BP	BP	1
18pF	Condensateur	C1, C3	0805	Condensateur	2
100nF	Condensateur	C2, C6, C9	0805	Condensateur	3
7pF	Condensateur	C4, C5	0805	Condensateur	2
1μF	Condensateur	C7, C8	0805	Condensateur	2
10 μF	Condensateur	C10	0805	Condensateur	1
100 nF	Condensateur	C11	0805	Condensateur	1
Diode	led	D1, D2	led	Diode	2
Header MCU->MCU		J1	Header 4	Header 4	1
Header IMU		J2	Header 8	Header	1
Header OPTIQUE		J3, J4	Header 8	Header	2
100R		R1	0805	Resistance	1
1K5		R2, R12	0805	Resistance	2
1M		R3, R4	0805	Resistance	2
100K		R5, R10	0805	Resistance	2
150R		R6, R13	0805	Resistance	2
47R		R7, R11	0805	Resistance	2
33R		R8, R9	0805	Resistance	2
22K		R14, R15	0805	Resistance	2
Transistor		T1	Transistor	Transistor	1
MC9S08JS16		U1	MC9S08JS16	MC9S08JS16	1
SN74LV125AD		U2	SN74LV125AD	SN74LV125AD	1
USB B		U3	USB_B	USB B	1
TC1262		U4	TC1262	TC1262	1
MC9S08QE16		U5	LQFP_32	MC9S08QE16	1
Quartz 12 MHz		Y1	Quartz	Quartz	1
Quartz 32.768khz		Y2	Quartz 32.768KHZ	Quartz 32.768khz	1

## Tests & validations

Pour la réalisation de la carte, n'étant pas très habile en soudure, j'ai demandé à un de mes camarades de la réaliser. Cependant lors de la première impression de la carte, après avoir réaliser les tests de continuité et ohmiques, lors du branchement de la carte aucun périphérique n'a été repérer sur le PC. Le programmeur a chauffé, complètement inutilisable. Je suis donc allé regarder ma schématique, et en effet j'ai inversé les broches au niveau de l'USB, les documentations sur les pins de l'USB ne sont pas indiquées dans toutes les documentations techniques mais j'aurai dû faire attention et vérifié plusieurs fois. J'ai du retiré une dernière carte qui elle fonctionnait en partie car le transistor ne jouait pas le rôle qu'il devait tenir. Pourtant, il commutait correctement après vérification à l'oscilloscope, mais j'ai dû le court-circuité et faire des reprises car là aussi j'ai eu à inverser le collecteur et l'émetteur. Une dernière reprise sur le programmeur, car j'ai branché la diode d'IHM sur la mauvaise broche.

Ces erreurs sont dû à mon manque d'expérience mais plus surtout à mon manque de vérification de ma schématique. J'ai pourtant eu à vérifier plusieurs fois et même fait vérifier la schématique par mes camarades mais c'était à moi de vérifier étape par étape. Je ferais plus attention à l'avenir.



## ETUDE LOGICIELLE

Afin de satisfaire mes différentes étapes logicielles, j'ai d'abord eu à réaliser la configuration des différentes fonctions que je vais utiliser à travers les microcontrôleurs.

### Fonction I<sup>2</sup>C

La norme I<sup>2</sup>C (inter-Integrated Circuit) est un bus série asynchrone bidirectionnel opérant half-duplex, où plusieurs maîtres et esclaves peuvent communiquer sur le bus au moyen d'adresses. Le bus nécessite deux lignes à savoir SDA (Serial data line) et SCL (Serial clock line) sans oublier la référence commune le ground. Les deux lignes sont tirées vers le haut grâce à une pull-up. Il a été développé par Phillips (maintenant NXP) en 1982. Depuis 2014 nous en sommes à la version 6 de la norme.

Procédons à l'étude de registre de la configuration du module I<sup>2</sup>C pour la communication avec l'IMU

Registre IICA :

AD7	AD6	AD6	AD3	AD2	AD1	0
-----	-----	-----	-----	-----	-----	---

Registre 7 bits contenant l'adresse de l'esclave, il est accessible en lecture et écriture.

Nous y mettrons l'adresse 0x68 correspondant à l'adresse de l'IMU

Registre IIC :

Ce registre configure la vitesse de communication du bus I<sup>2</sup>C.

MULT		ICR					
0	0	0	0	0	0	0	1

MULT [7-6] : Les deux derniers bits du registre sélectionne le facteur de multiplication de la vitesse de communication. Nous le mettrons à 1.

ICR [5-0] : Pour remplir ces bits, il faut procéder au calcul de la vitesse de communication qui est indiquée dans la documentation construction du microcontrôleur :

$$Speedbus = \frac{F_{bus}}{MULT * ICR}$$

$$ICR = \frac{25.165 * 10^6}{400 * 10^3} = 62.91$$

Il faudra ensuite se référer au table 12-4 dans le chapitre consacré à IIC pour choisir la valeur du registre ICR, j'ai donc choisi la valeur 0x12 car elle correspondait à 64, la valeur se rapprochant le plus du mon résultat.

J'ai donc procédé au calcul inverse pour savoir à quelle fréquence sera exactement le bus I<sup>2</sup>C.

$$Speedbus = \frac{25.165 * 10^6}{64} = 393,2 \text{ kHz}$$

Calcul du taux d'erreur :

$$Te = 1 - \frac{393.2}{400} * 100 = 1.7\%$$

Registre IIC1

IICEN		MST	TX	TXAK	0	0	0
1		1			RSTA	0	0

IICEN [7] : A 1 active la fonction I<sup>2</sup>C.

MST [6] : A 1 configure le microcontrôleur en maître.

TX [5] : A 0 configure la fonction en réception, à 1 configure la fonction en transmission

TXAK [4] : A 0 le microcontrôleur transmet un signal d'acquittement en réponse à une réception d'un octet. A 1, il transmet un signal de non acquittement pour interrompre la communication.

RSTA [3] : Une écrire dans ce bit, indique au microcontrôleur de générer une condition de repeat start.

Les autres bits ne feront pas parti de l'étude, ils seront donc laissés dans leur état par défaut.

Registre IICS :

TCF		BUSY					RXAK

TCF [7] : A 1, ce flag indique la fin d'une transmission ou d'une réception. Il se met à zéro après avoir été lu et suivant une écriture dans IICD.

BUSY [5] : A 1, ce flag indique si le bus détecte un signal de start et à zéro lorsqu'il détecte un signal de stop.

RXAKA [0] : A 0, ce bit détecte si le signal d'acquittement a été reçu, à 1 si c'est un signal de non acquittement.

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

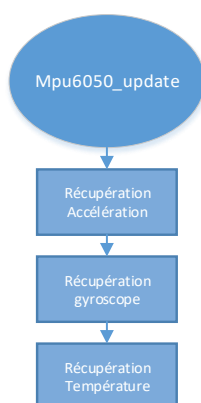
Registre IICD :

Data
------

Ce registre 8 bits contient les données à transmettre ou celles reçues.

Concernant la programmation du bus I<sup>2</sup>C, j'ai généré les fonctions d'initialisation, d'émission et de réception via au processeur expert de CodeWarrior. Elles ne feront donc pas l'objet de représentation par algorithme.

Fonction du capteur :



## Fonction SPI

La liaison SPI (Serial Peripheral Interface) est un bus série synchrone opérant en full-duplex, où un maître et un esclave se parle à transfert les signaux MOSI (Master Output Slave Input), MISO (Master Input Slave Output), SCLK (Serial Clock) et SS (Slave Select). Ils ont besoin également du ground comme référence. Le bus SPI est basé sur le principe de registre à décalage. Lorsque le maître envoie des commandes sur la ligne MOSI, ce n'est qu'au prochain coup d'horloge que l'esclave répondra sur la ligne MISO. Pendant ce temps, la ligne SS doit être tirée à l'état bas pour faire comprendre à l'esclave qu'il y a un maître présent. S'il y a plus d'esclaves, ils doivent tous avoir un SS différent.

Procédons à l'étude de registre :

Registre SPIC1 :

SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
	1		1	1	1	0	0

SPE [6] : A 1 active la fonction SPI

MSTR [4] : A 1 configure le microcontrôleur en maître

CPOL [3] : A 1 définit l'état de repos de l'horloge à l'état haut

CPHA [2] : A 1 l'acquisition des données se fera sur le front du milieu

SSOE [1] : Bit est utilisé en combinaison avec le bit MODFEN du registre SPIC2 pour définir le mode de fonctionnement du chip select, nous le mettrons à 0 ;

LSBFE [0] : A 1 lors de la transmission des données, le premier bit à être envoyé sera le MSB.

Les autres bits ne feront pas parti de notre étude.

Registre SPIC2 :

0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPCO
			0				

MODFEN [4] : Nous mettrons ce bit à 0 pour configurer la broche chip select comme I/O, cela nous permettra d'avoir plusieurs esclaves sur le bus SPI.

Les autres bits seront laissés dans leur état par défaut.

Registre SPIBR :

0	SPPR2	SPPR1	SPPR0	SPR3	SPR2	SPR1	SPR0
	0	0	0	0	0	1	0

SPPR [6-4] : Ces bits permettent une première division de l'horloge du bus. Pour connaître la division, il faut se référer à la table 15-6 du chapitre réservé à la SPI.

SPR [3-0] : Ces bits permettent la division de l'horloge du bus. Notre horloge étant de 25MHz, nous allons donc diviser par 8 pour que la fréquence de notre horloge soit de 3 MHz.

Registre SPIS

SPRF	0	SPTEF	MODF	0	0	0	0

SPRF [7] : Indicateur de fin de transmission, à 1 lorsqu'une donnée est disponible dans SPID, remise à zéro par lecture du registre SPIS lorsque SPRF est à 1 suivi d'une lecture dans SPID.

SPTEF [5] : Indicateur de fin de transmission, à 1 lorsque le buffer de transmission est vide, remise à zéro par lecture du registre SPIS lorsque SPTEF est à 1 suivi d'une écriture dans SPID.

Les autres bits seront laissés dans leur état par défaut.

Registre SPID :

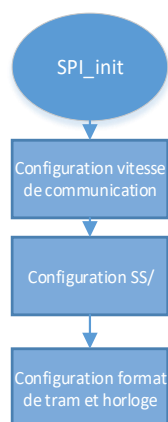
Data
------

Ce registre 8 bits contient les données à transmettre ou celles reçues.

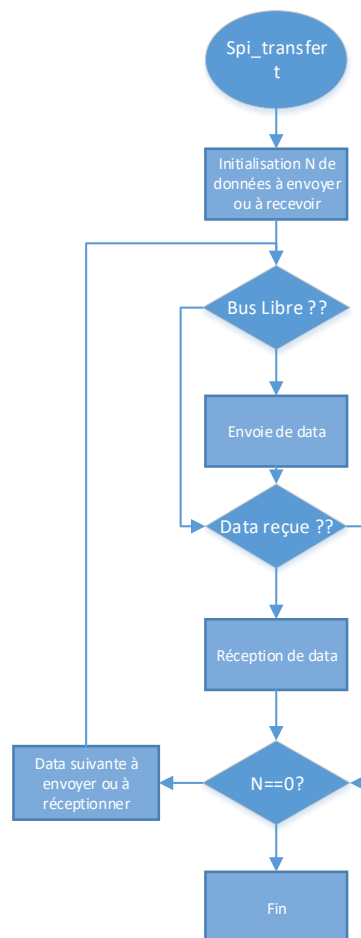
La fonction SPI m'a permis de mettre en œuvre les capteurs optiques, voici donc les organigrammes des programmes que j'ai utilisé.

Organigramme :

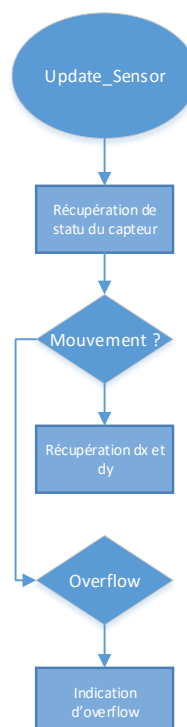
Initialisation :



Mise en œuvre :



Fonction du capteur :



## Fonction UART

L'UART (Universal Asynchronous Receiver Transmitter) est le mode de communication pour relier un pc et un port série. Une trame UART se compose de la suite :

- Un bit de start toujours à zéro.
- Les données, une trame de 5 à 9 bits commençant toujours par le LSB.
- La parité, paire ou impaire.
- Un bit de stop toujours à 1, la durée du bit peut varier de 1 à 2.

Registre SCIBD

			SBR12	SBR11	SBR10	SBR9	SBR8

SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0

SBR [12-0] : Ces bits permettent de déterminer la vitesse de communication. Pour les remplir, la documentation du constructeur nous donne une formule à utiliser.

$$Baudrate = \frac{F_{bus}}{SBR[12 - 0] * 16}$$

$$SBR[12 - 0] = \frac{25.165 * 10^6}{115200 * 16} = 13.65$$

Réalisons le calcul inverse pour calculer le taux d'erreur :

$$Baudrate = \frac{25.165 * 10^6}{14 * 16} = 112343.75$$

$$Te = 1 - \frac{112343.75}{115200} = 2.48 \%$$

Registre SCI1C1

			M			PE	PT
			0			0	

M [4] : A 0 nous configurons le format de trame en 8 bits de données et 1 bit de stop.

PE [1] : A 0 pas de parité.

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

Registre SCI1C2

TIE	TCIE	RIE		TE	RE		
-----	------	-----	--	----	----	--	--

		1		1	1		
--	--	---	--	---	---	--	--

RIE [5] : A 1 une interruption se déclenche lorsque le bit RDRF du registre SCIC1S1 est à 1.

TE [3] : A 1 active la transmission du module UART

RE [2] : A 1 active la réception du module UART

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

Registre SC11S1 :

TDRE		RDRF					
------	--	------	--	--	--	--	--

TDRE [7] : Indicateur de fin de transmission, à 1 lorsque le buffer de transmission est vide, remise à zéro par lecture du registre SC11S1 lorsque TDRE est à 1 suivi d'une écriture dans SC11D.

RDRF [7] : Indicateur de fin de transmission, à 1 lorsque le buffer de réception est plein, remise à zéro par lecture du registre SC11S1 lorsque RDRF est à 1 suivi d'une lecture dans SC11D.

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

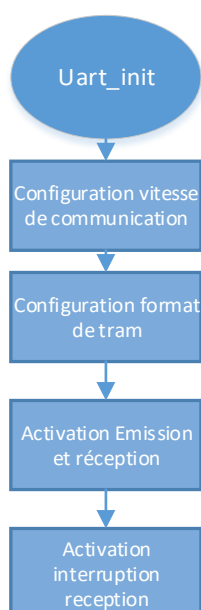
Registre SC11D :

R7	R6	R5	R4	R3	R2	R1	R0
T7	T6	T5	T4	T3	T2	T1	T0

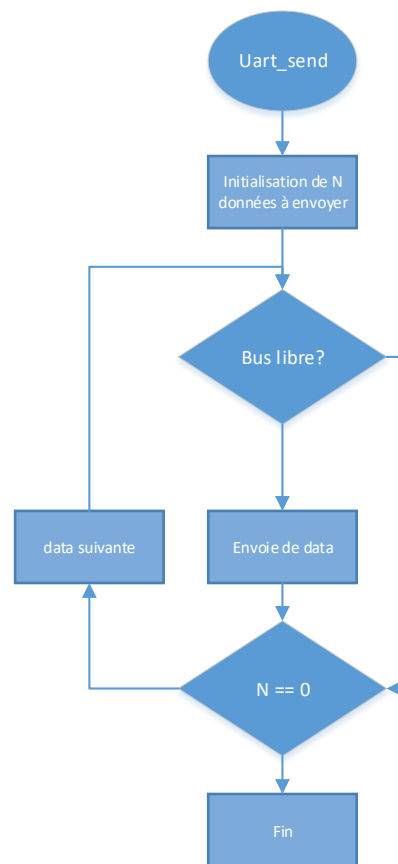
Ce registre 8 bits contient les données à transmettre ou celles reçues.

Organigramme :

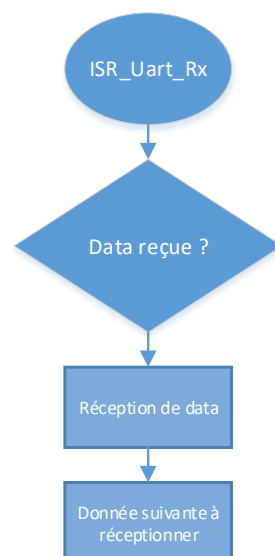
Initialisation :



Emission :



Réception :





## Fonction Timer

La fonction timer du microcontrôleur regroupe plusieurs sous-fonctions à savoir la PWM (Pulse With Modulation), Output Compare et Input Capture. Pour la réalisation de certains calculs, je vais mettre en œuvre plusieurs interruptions timers et configurer les PWM pour gérer les vitesses et les déplacements du robot.

### PWM

Pour donner des ordres au ESC (Electronic Speed Control), il faut une fréquence entre 20 et 30kHz. Je vais donc présenter l'étude de registre appropriée pour les contrôler.

Registre TPMxSC :

TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
		0	0	1	0	0	0

CPWMS [5] : A 0 le channel opéra en mode PWM, Output compare ou input capture.

CLKS(B-A) [4-3] : A 01 l'horloge de la PWM sera fixée par la fréquence de bus.

PS [2-0] : A 000 il n'y a pas de division de l'horloge.

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

Registre TPMOD :

TPMxMOD [15-0]
----------------

Ce double registre 8 bits contiendra la valeur de la période du signal.

Procédons au calcul de la valeur à y mettre pour 20KHz.

$$\text{Nombre de timerticks} = \frac{1}{25.165 * 10^6} = 39.73 \text{ ns}$$

1 timertick correspondra donc à 39.73 ns. Nous allons nous servir de cela pour calculer le nombre de timertick correspondant à 20 kHz.

$$N = \frac{50 \mu s}{39.73 \text{ ns}} = 1258 (10) = 0x04E9(16)$$

Registre TPMxCxSC :

CHnF	CHnIE	MSnB	MSnA	ELSnA	ELSnB	0	0
		1	1	1	1		

MSn [A-B] : A 11 nous définissons une configuration Edge-aligned PWM.

ELSn[A-B] : A 11 nous définissons un état de repos bas pour la PWM.

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

Registre TPMxCnV :

TPMxCnV [15-0]
----------------

Ce double registre 8 bits contiendra la valeur du rapport cycle à appliquer à chaque ESC.

La configuration des 4 channels sera la même pour chaque ESC.

### Output Compare

Afin de rafraîchir à intervalle fixe les données brutes des capteurs, il faut mettre en œuvre un certain nombre d'interruption timer.

Registre TPMxSC :

TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
	1	0	0	1	1	1	1

TOF [7] : Indicateur de débordement du compteur, à 1 lorsque le compteur dépasse sa capacité de comptage, remise à zéro par lecture de TPMxSC lorsque ce bit est mis à 0.

TOIE [6] : A 1 ce bit active une interruption lorsque le bit TOF est à 1.

CPWMS [5] : À 0 ce bit permet de choisir trois fonctions du timer Output compare, input capture et PWM

CLKS [4 :3] : A 01 ces deux bits sélectionnent l'horloge de bus comme horloge du compteur.

PS [2 :0] : A 111 divise l'horloge du compteur par 128.

Registre TPMOD :

TPMxMOD [15-0]
----------------

Ce double registre 8 bits contiendra la valeur de la période du signal.

Procédons au calcul de la valeur à y mettre pour 20 ms.

$$\text{Nombre de timerticks} = \frac{128}{25.165 * 10^6} = 5.08 \mu s$$

1 timertick correspondra donc à 39.73 ns. Nous allons nous servir de cela pour calculer le nombre de timertick correspondant à 20 ms.

$$N = \frac{20 \text{ ms}}{39.73 \text{ ns}} = 3932 (10) = 0x0F5C(16)$$

Registre TPMxCxSC :

CHnF	CHnIE	MSnB	MSnA	ELSnA	ELSnB	0	0
		0	1	0	0		

MSn [A-B] : A 01 configure le timer en Output compare.

ELSn[A-B] : A 00 configure l'Output compare en compteur logiciel.

Les autres bits ne feront pas partie de l'étude, ils seront donc laissés dans leur état par défaut.

Les procédures d'initialisations des fonctions PWM et Output Compare se feront via le processeur expert de l'IDE.

## Asservissement en vitesse

Pour assurer un contrôle optimal du robot lors de ses déplacements, il faut mettre en place un régulateur PID.

Un régulateur PID permet d'améliorer grandement l'asservissement d'une donnée.

Explication du régulateur :

Composante proportionnelle :

Il s'agit d'une correction instantanée de l'erreur entre la mesure et la consigne, pour être rapidement en mesure d'atteindre la consigne et que le système réagisse aux changements de consignes, c'est un coefficient en général nommé  $k_p$ . Cependant ce coefficient seul ne suffit pas, car plus l'erreur est grande plus le coefficient de correction devra être grand, ce qui peut rendre le système instable.

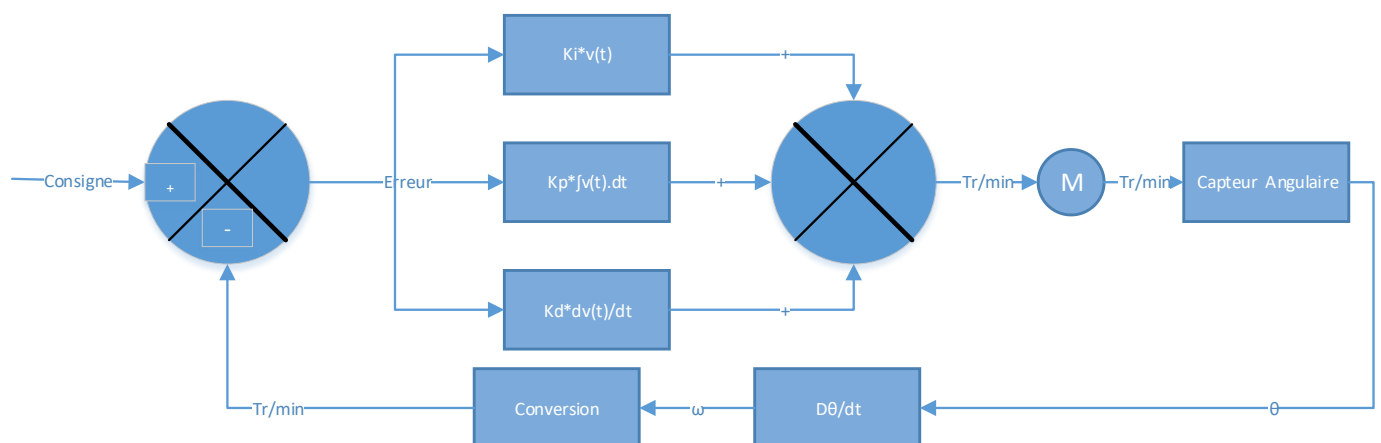
Composante intégrale :

Il s'agit de faire la somme des erreurs mesurées et l'ajouter à la précédente composante, pour combler l'erreur dite statique. En effet, quand le système s'approchera de la consigne, il restera cependant une erreur qui pourra empêcher le système d'atteindre la consigne ou peut la dépasser. Lorsque le système atteindra la consigne, la composante deviendra constante car l'erreur sera nulle. Elle permettra donc d'obtenir un système stable. Cependant elle ne réagit pas bien aux perturbations, donc lorsqu'un changement important de la consigne apparaît, le système se met à osciller. Cette composante est multipliée par un coefficient nommé  $k_i$  qui permet d'augmenter l'effet de la composante pour réduire l'erreur statique.

Composante Dérivée :

Cette fois-ci, il s'agit de faire la soustraction de l'erreur entre la consigne et la mesure, pour empêcher un dépassement de la consigne à atteindre. Cette composante possède aussi un coefficient nommé  $k_d$  qui permet d'augmenter l'effet de la composante pour réduire le dépassement.

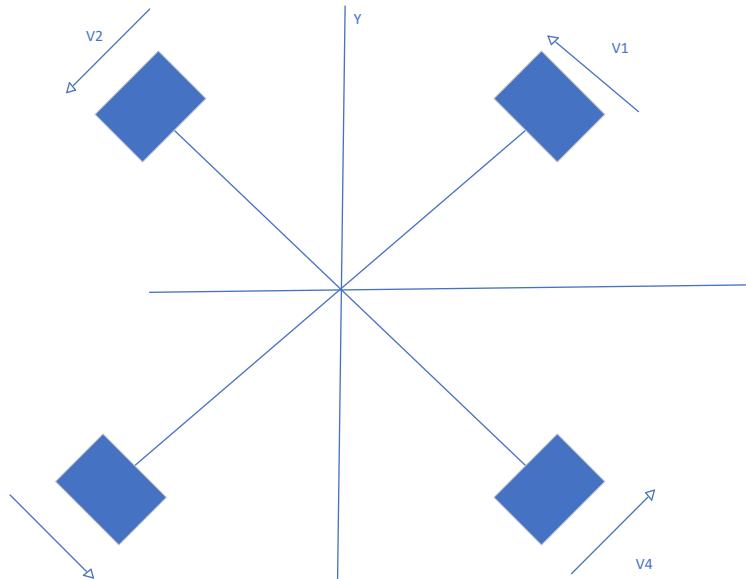
Schéma d'asservissement :



## Déplacement & Position

### Détermination des équations de déplacements

Pour le calcul des déplacements, je me suis heurté à un problème de puissance de calcul au niveau du microcontrôleur. En effet les calculs utilisant la librairie `<math.h>` n'était pas envisageable. Le microcontrôleur ne supportait pas ce type de calcul. Notre intervenant m'a alors conseillé d'utiliser le produit scalaire pour calculer les déplacements du robot ainsi que la vitesse nécessaire à chaque roue pour réaliser le déplacement. Le principe est simple, chaque moteur à un vecteur fixe qui ne changera jamais sauf si on change la disposition du robot. J'ai donc demandé à mon camarade en charge du châssis de me donner la disposition des roues. Ainsi je pouvais déterminer les angles pour le calcul des vecteurs.



Calcul des composantes x et y de chaque vecteur :

Moteur 1 :

$$x_{m1} = \cos\left(21,5 + \frac{\pi}{2}\right) = -0,365$$

$$y_{m1} = \sin\left(21,5 + \frac{\pi}{2}\right) = 0,930$$

Moteur 2 :

$$x_{m2} = \cos\left(158,5 + \frac{\pi}{2}\right) = -0,368$$

$$y_{m2} = \sin\left(158,5 + \frac{\pi}{2}\right) = -0,929$$

Moteur 3 :

$$x_{m3} = \cos\left(225 + \frac{\pi}{2}\right) = 0,705$$

$$y_{m3} = \sin\left(225 + \frac{\pi}{2}\right) = -0,709$$

Moteur 4 :

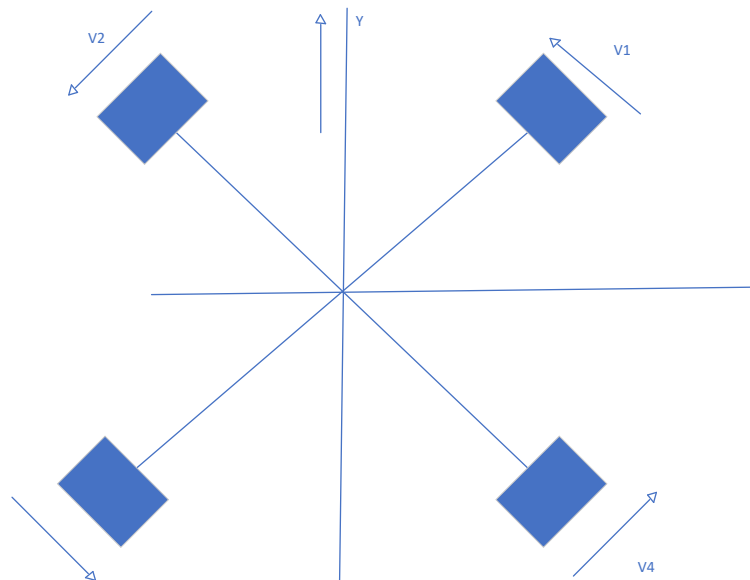
$$x_{m4} = \cos\left(-45 + \frac{\pi}{2}\right) = 0,707$$

$$y_{m4} = \sin\left(-45 + \frac{\pi}{2}\right) = 0,706$$

Maintenant que nous avons les vecteurs des moteurs, nous allons les multiplier à un vecteur direction, additionner les composantes dx et dy pour former un vecteur puis transformer en tour/min pour connaître la vitesse à laquelle la roue doit tourner pour effectuer le mouvement. Le signe + ou – indiquera le sens de rotation de la roue.

Procédons à un exemple :

Nous prenons l'exemple ci-dessous, le robot doit se déplacer en avant :



Nous considérons une longueur de 1 pour le vecteur.

Multiplication du vecteur direction avec les vecteurs des moteurs :

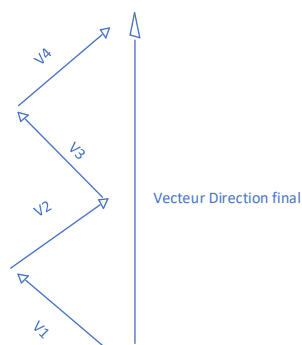
Moteur 1 :  $dx = 0 ; dy = 1 * y_{m1} ; dv_1 = dx + dy = 0,93$

Moteur 2 :  $dx = 0 ; dy = 1 * y_{m2} ; dv_2 = dx + dy = -0,92$

Moteur 3 :  $dx = 0 ; dy = 1 * y_{m3} ; dv_3 = dx + dy = -0,709$

Moteur 4 :  $dx = 0 ; dy = 1 * y_{m4} ; dv_4 = dx + dy = 0,706$

On remarque déjà que chaque moteur opposé à l'axe y tournera sensible à la même vitesse mais pas dans la même direction. Faisons un peu l'addition de vecteur pour voir la forme de notre direction :



Il suffira ensuite de convertir les transformer en tour par min. Mais avant ça, il faut définir une vitesse. Admettons que nous nous déplaçons à 3 m/s, en tenant compte du diamètre de la roue. Le diamètre de nos roues est de 0,06m

$$\text{Moteur 1 : } V_1 = \frac{60 * dv_1 * 3}{0,06 * 3,14} = 1482 \text{ tr/min}$$

$$\text{Moteur 2 : } V_2 = \frac{60 * dv_2 * 3}{0,06 * 3,14} = -1482 \text{ tr/min}$$

$$\text{Moteur 3 : } V_3 = \frac{60 * dv_3 * 3}{0,06 * 3,14} = -1129 \text{ tr/min}$$

$$\text{Moteur 4 : } V_4 = \frac{60 * dv_4 * 3}{0,06 * 3,14} = 1125 \text{ tr/min}$$

Nous avons ainsi les différentes valeurs à appliquer au PWM de chaque moteur et leur sens de rotation.

### Calcul de la position du robot

Pour calculer la position courante du robot, je possède de nombreux capteurs pour le faire.

Centrale inertielle :

Pour cette dernière, il existe beaucoup d'exemple de calcul de position Sachant cela j'ai cherché d'autres solutions qui elles sont toutes aussi couteuses en performance. J'ai donc décidé de déporter ses calculs vers le PC.

Utilisation d'un filtre complémentaire :

On calcul les angles d'inclinaisons de l'accéléromètre sur chaque axe, on calcul les angles à partir du gyroscope sur chaque axe puis on fait de la fusion de données pour compenser les erreurs de mesure de chaque capteur en appliquant un filtre à chaque capteur. Un filtre passe haut pour le gyroscope car il réagit mieux aux variations rapides, un filtre passe bas pour l'accéléromètre car il réagit mieux aux variations lentes.

La formule du filtre complémentaire appliquer aux capteurs est la suivante :

$$angle = a * (angle(t - 1) + anglegyroscope) + ((1 - a) * angleaccéléromètre)$$

Où  $a$  représente la constante caractéristique des filtres qui se détermine par la formule :

$$a = \left(\frac{1}{fc}\right) / \left(Te + \frac{1}{fc}\right)$$

Nous allons couper le plus bas possible à 4Hz et notre temps d'échantillonnage a été défini lors de l'étude de la fonction timer, il sera donc de 20 ms.

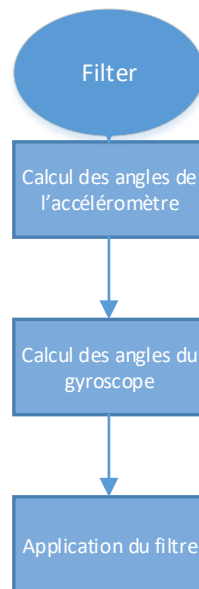
$$a = 0.0123$$

La formule utilisée dans notre programme sera alors :

$$angle = 0.0123 * (angle(t - 1) + anglegyroscope) + ((0.9876) * angleaccéléromètre)$$

Ce qu'il me manquera c'est interpréter correctement les angles pour déterminer exactement où le robot se trouve.

Algorithme :



Capteur angulaire :

Odométrie :

Pour effectuer l'odométrie du robot, il faut utiliser les angles retournés puis les intégrer en vitesse de rotation. Ensuite il faut calculer la vitesse du robot pour déterminer la distance parcourue et le rayon de courbure du robot pour déterminer l'angle de rotation du robot.

$$v = \frac{V_1 + V_2 + V_3 + V_4}{4}$$

$$R = \frac{e}{4} * \frac{V_1 + V_2 + V_3 + V_4}{V_1 + V_2 - V_3 + V_4}$$

$$d\theta = \frac{\int d \cdot dt}{R}$$

Calcul de la position :

$$x = R \cos(\theta) + x_0$$

$$y = R \sin(\theta) + y_0$$

Malheureusement je n'ai pas pu tester la mise en œuvre logicielle de ma solution car je n'ai pas reçu les capteurs angulaires.

## CONCLUSION

Au cours de ce projet, j'ai eu à réaliser beaucoup de recherches pour comprendre comment me servir efficacement des capteurs. En particulier pour la centrale inertielle, cette dernière permet de faire énormément de calculs si la puissance de calcul est au rendez-vous. Ce qui malheureusement à cause de l'architecture de notre famille de microcontrôleur n'est pas possible. J'ai essayé plusieurs types de calculs mais seuls les calculs à virgules fixes sont possibles et quelques calculs avec des floats aussi mais sachant que les opérations à faire demande de la précision, on ne peut se permettre de négliger cela. Nous n'avons pas pu mener le projet à bien. En effet, la partie logicielle concernant le PID le calcul de position et de déplacements ne sont que théoriques, je n'ai pas pu mener de tests car le robot n'a pas pu voir le jour. En ce qui concerne le capteur angulaire malheureusement je n'ai pas pu le tester, nous le voulions en sortie i<sup>2</sup>c mais le temps de réapprovisionnement était plus long que prévu et jusqu'alors nous ne l'avons toujours pas reçu. Si le projet était à poursuivre, il faudrait changer d'architecture de microcontrôleur et placé un microcontrôleur sur chaque ESC car une seule unité de traitement ne pourrait pas supporter tous les calculs sauf s'ils sont déportés encore sur PC.

## ANNEXE

Etant un grand passionné d'open source et utilisant beaucoup le gestionnaire de version git, vous trouverez l'intégralité des programmes réalisés sur mon github. La lisibilité du code étant meilleur également, il vous sera plus agréable pour comprendre les programmes. Un README détaillé vous expliquera comment naviguer dans le dépôt. Voici le lien : [Projet Robocup](#).